CLASSIFICATION OF PIXELS IN A MICROARRAY IMAGE BASED ON PIXEL INTENSITIES AND A PREVIEW MODE FACILITATED BY PIXEL-INTENSITY-BASED PIXEL CLASSIFICATION

TECHNICAL FIELD

The present invention is related to processing of microarray data and, in particular, to a method and system for partitioning pixels in an image of a microarray into a set of feature pixels and a set of background pixels.

BACKGROUND OF THE INVENTION

The present invention is related to methods and systems for determining which pixels, in a digital image of a microarray, are associated with features of the microarray, and which pixels are background pixels associated with inter-feature regions of a microarray. A general background of microarray technology is first provided, in this section, to facilitate discussion of microarray-data processing, in following subsections. It should be noted that microarrays are also referred to as "microarrays" and simply as "arrays." These alternate terms may be used interchangeably in the context of microarrays and microarray technologies. Art described in this section is not admitted to be prior art to this application.

Array technologies have gained prominence in biological research and are likely to become important and widely used diagnostic tools in the healthcare industry. Currently, microarray techniques are most often used to determine the concentrations of particular nucleic-acid polymers in complex sample solutions. Molecular-array-based analytical techniques are not, however, restricted to analysis of nucleic acid solutions, but may be employed to analyze complex solutions of any type of molecule that can be optically or radiometrically scanned and that can bind with high specificity to complementary molecules synthesized within, or bound to, discrete features on the surface of an array. Because arrays are widely used for analysis of nucleic acid samples, the following background information on arrays is introduced in the context of analysis of nucleic acid solutions following a brief background of nucleic acid chemistry.

Deoxyribonucleic acid ("DNA") and ribonucleic acid ("RNA") are linear polymers, each synthesized from four different types of subunit molecules. The subunit

molecules for DNA include: (1) deoxy-adenosine, abbreviated "A," a purine nucleoside; (2) deoxy-thymidine, abbreviated "T," a pyrimidine nucleoside; (3) deoxy-cytosine, abbreviated "C," a pyrimidine nucleoside; and (4) deoxy-guanosine, abbreviated "G," a purine nucleoside. The subunit molecules for RNA include: (1) adenosine, abbreviated "A," a purine nucleoside; (2) uracil, abbreviated "U," a pyrimidine nucleoside; (3) cytosine, abbreviated "C," a pyrimidine nucleoside; and (4) guanosine, abbreviated "G," a purine nucleoside. Figure 1 illustrates a short DNA polymer 100, called an oligomer, composed of the following subunits: (1) deoxy-adenosine 102; (2) deoxy-thymidine 104; (3) deoxy-cytosine 106; and (4) deoxy-guanosine 108. When phosphorylated, subunits of DNA and RNA molecules are called "nucleotides" and are linked together through phosphodiester bonds 110-115 to form DNA and RNA polymers. A linear DNA molecule, such as the oligomer shown in Figure 1, has a 5' end 118 and a 3' end 120. A DNA polymer can be chemically characterized by writing, in sequence from the 5' end to the 3' end, the single letter abbreviations for the nucleotide subunits that together compose the DNA polymer. For example, the oligomer 100 shown in Figure 1 can be chemically represented as "ATCG." A DNA nucleotide comprises a purine or pyrimidine base (e.g. adenine 122 of the deoxy-adenylate nucleotide 102), a deoxy-ribose sugar (e.g. deoxy-ribose 124 of the deoxy-adenylate nucleotide 102), and a phosphate group (e.g. phosphate 126) that links one nucleotide to another nucleotide in the DNA polymer. In RNA polymers, the nucleotides contain ribose sugars rather than deoxy-ribose sugars. In ribose, a hydroxyl group takes the place of the 2' hydrogen 128 in a DNA nucleotide. RNA polymers contain uridine nucleosides rather than the deoxy-thymidine nucleosides contained in DNA. The pyrimidine base uracil lacks a methyl group (130 in Figure 1) contained in the pyrimidine base thymine of deoxy-thymidine.

The DNA polymers that contain the organization information for living organisms occur in the nuclei of cells in pairs, forming double-stranded DNA helixes. One polymer of the pair is laid out in a 5' to 3' direction, and the other polymer of the pair is laid out in a 3' to 5' direction. The two DNA polymers in a double-stranded DNA helix are therefore described as being anti-parallel. The two DNA polymers, or strands, within a double-stranded DNA helix are bound to each other through attractive forces including hydrophobic interactions between stacked purine and pyrimidine bases and hydrogen bonding between purine and pyrimidine bases, the attractive forces emphasized by conformational

constraints of DNA polymers. Because of a number of chemical and topographic constraints, double-stranded DNA helices are most stable when deoxy-adenylate subunits of one strand hydrogen bond to deoxy-thymidylate subunits of the other strand, and deoxy-guanylate subunits of one strand hydrogen bond to corresponding deoxy-cytidilate subunits of the other strand.

Figures 2A-B illustrates the hydrogen bonding between the purine and pyrimidine bases of two anti-parallel DNA strands. Figure 2A shows hydrogen bonding between adenine and thymine bases of corresponding adenosine and thymidine subunits, and Figure 2B shows hydrogen bonding between guanine and cytosine bases of corresponding guanosine and cytosine subunits. Note that there are two hydrogen bonds 202 and 203 in the adenine/thymine base pair, and three hydrogen bonds 204-206 in the guanosine/cytosine base pair, as a result of which GC base pairs contribute greater thermodynamic stability to DNA duplexes than AT base pairs. AT and GC base pairs, illustrated in Figures 2A-B, are known as Watson-Crick ("WC") base pairs.

Two DNA strands linked together by hydrogen bonds forms the familiar helix structure of a double-stranded DNA helix. Figure 3 illustrates a short section of a DNA double helix 300 comprising a first strand 302 and a second, anti-parallel strand 304. The ribbon-like strands in Figure 3 represent the deoxyribose and phosphate backbones of the two anti-parallel strands, with hydrogen-bonding purine and pyrimidine base pairs, such as base pair 306, interconnecting the two strands. Deoxy-guanylate subunits of one strand are generally paired with deoxy-cytidilate subunits from the other strand, and deoxy-thymidilate subunits in one strand are generally paired with deoxy-adenylate subunits from the other strand. However, non-WC base pairings may occur within double-stranded DNA.

Double-stranded DNA may be denatured, or converted into single stranded DNA, by changing the ionic strength of the solution containing the double-stranded DNA or by raising the temperature of the solution. Single-stranded DNA polymers may be renatured, or converted back into DNA duplexes, by reversing the denaturing conditions, for example by lowering the temperature of the solution containing complementary single-stranded DNA polymers. During renaturing or hybridization, complementary bases of anti-parallel DNA strands form WC base pairs in a cooperative fashion, leading to reannealing of the DNA duplex. Strictly A-T and G-C complementarity between anti-parallel polymers leads to the

greatest thermodynamic stability, but partial complementarity including non-WC base pairing may also occur to produce relatively stable associations between partially-complementary polymers. In general, the longer the regions of consecutive WC base pairing between two nucleic acid polymers, the greater the stability of hybridization between the two polymers under renaturing conditions.

The ability to denature and renature double-stranded DNA has led to the development of many extremely powerful and discriminating assay technologies for identifying the presence of DNA and RNA polymers having particular base sequences or containing particular base subsequences within complex mixtures of different nucleic acid polymers, other biopolymers, and inorganic and organic chemical compounds. One such methodology is the array-based hybridization assay. Figures 4-7 illustrate the principle of the array-based hybridization assay. An array (402 in Figure 4) comprises a substrate upon which a regular pattern of features is prepared by various manufacturing processes. The array 402 in Figure 4, and in subsequent Figures 5-7, has a grid-like 2-dimensional pattern of square features, such as feature 404 shown in the upper left-hand corner of the array. Each feature of the array contains a large number of identical oligonucleotides covalently bound to the surface of the feature. These bound oligonucleotides are known as probes. In general, chemically distinct probes are bound to the different features of an array, so that each feature corresponds to a particular nucleotide sequence. In Figures 4-6, the principle of array-based hybridization assays is illustrated with respect to the single feature 404 to which a number of identical probes 405-409 are bound. In practice, each feature of the array contains a high density of such probes but, for the sake of clarity, only a subset of these are shown in Figures 4-6.

Once an array has been prepared, the array may be exposed to a sample solution of target DNA or RNA molecules (410-413 in Figure 4) labeled with fluorophores, chemiluminescent compounds, or radioactive atoms 415-418. Labeled target DNA or RNA hybridizes through base pairing interactions to the complementary probe DNA, synthesized on the surface of the array. Figure 5 shows a number of such target molecules 502-504 hybridized to complementary probes 505-507, which are in turn bound to the surface of the array 402. Targets, such as labeled DNA molecules 508 and 509, that do not contains nucleotide sequences complementary to any of the probes bound to array surface do not

hybridize to generate stable duplexes and, as a result, tend to remain in solution. The sample solution is then rinsed from the surface of the array, washing away any unbound-labeled DNA molecules. In other embodiments, unlabeled target sample is allowed to hybridize with the array first. Typically, such a target sample has been modified with a chemical moiety that will react with a second chemical moiety in subsequent steps. Then, either before or after a wash step, a solution containing the second chemical moiety bound to a label is reacted with the target on the array. After washing, the array is ready for scanning. Biotin and avidin represent an example of a pair of chemical moieties that can be utilized for such steps.

Finally, as shown in Figure 6, the bound labeled DNA molecules are detected via optical or radiometric scanning. Optical scanning involves exciting labels of bound labeled DNA molecules with electromagnetic radiation of appropriate frequency and detecting fluorescent emissions from the labels, or detecting light emitted from chemiluminescent labels. When radioisotope labels are employed, radiometric scanning can be used to detect the signal emitted from the hybridized features. Additional types of signals are also possible, including electrical signals generated by electrical properties of bound target molecules, magnetic properties of bound target molecules, and other such physical properties of bound target molecules that can produce a detectable signal. Optical, radiometric, or other types of scanning produce an analog or digital representation of the array as shown in Figure 7, with features to which labeled target molecules are hybridized similar to 706 optically or digitally differentiated from those features to which no labeled DNA molecules are bound. In other words, the analog or digital representation of a scanned array displays positive signals for features to which labeled DNA molecules are hybridized and displays negative features to which no, or an undetectably small number of, labeled DNA molecules are bound. Features displaying positive signals in the analog or digital representation indicate the presence of DNA molecules with complementary nucleotide sequences in the original sample solution. Moreover, the signal intensity produced by a feature is generally related to the amount of labeled DNA bound to the feature, in turn related to the concentration, in the sample to which the array was exposed, of labeled DNA complementary to the oligonucleotide within the feature.

One, two, or more than two data subsets within a data set can be obtained from a single microarray by scanning the microarray for one, two or more than two types of signals.

Two or more data subsets can also be obtained by combining data from two different arrays. When optical scanning is used to detect fluorescent or chemiluminescent emission from chromophore labels, a first set of signals, or data subset, may be generated by scanning the microarray at a first optical wavelength, a second set of signals, or data subset, may be generated by scanning the microarray at a second optical wavelength, and additional sets of signals may be generated by scanning the molecular at additional optical wavelengths. Different signals may be obtained from a microarray by radiometric scanning to detect radioactive emissions one, two, or more than two different energy levels. Target molecules may be labeled with either a first chromophore that emits light at a first wavelength, or a second chromophore that emits light at a second wavelength. Following hybridization, the microarray can be scanned at the first wavelength to detect target molecules, labeled with the first chromophore, hybridized to features of the microarray, and can then be scanned at the second wavelength to detect target molecules, labeled with the second chromophore, hybridized to the features of the microarray. In one common microarray system, the first chromophore emits light at a red visible-light wavelength, and the second chromophore emits light at a green, visible-light wavelength. The data set obtained from scanning the microarray at the red wavelength is referred to as the "red signal," and the data set obtained from scanning the microarray at the green wavelength is referred to as the "green signal." While it is common to use one or two different chromophores, it is possible to use one, three, four, or more than four different chromophores and to scan a microarray at one, three, four, or more than four wavelengths to produce one, three, four, or more than four data sets.

In a raw and/or partially processed scanned image of a microarray, features may be asymmetrical, may contain highly non-uniform intensities due to a large number of different possible procedural, experimental, and instrumental errors and instabilities, and may be substantially offset from their expected positions within the general grid-like, regular pattern in which features are deposited. All of these effects can lead to difficulties in employing the currently used, intensity-based feature-pixel/background-pixel partitioning methods described above. For these reasons, there is a need for other methods for partitioning pixels within a digital image of a microarray into sets of feature pixels and background pixels.

SUMMARY OF THE INVENTION

In one embodiment of the present invention, a method based on an iteratively employed Bayesian-probability-based pixel classification is used to refine an initial feature mask that specifies those pixels in a region of interest, including and surrounding a feature in the scanned image of a microarray, that together compose a pixel-based image of the feature within the region of interest. In the described embodiment, a two-dimensional Boolean array is employed as a feature mask, each cell within the two-dimensional array indicating whether or a corresponding pixel within the pixel-based scanned image of the region of interest surrounding and including the feature is a feature pixel or a background pixel. The feature mask is prepared using the pixel-based intensity data for a region of interest, a putative position and size of the feature within the region of interest, and mathematical models of the probability distribution of background-pixel and feature-pixel signal noise and mathematical models of the probabilities of finding feature pixels and background pixels at various distances from the putative feature position. Preparation of a feature mask allows a feature-extraction system to display feature sizes and locations to a user prior to undertaking the computationally intensive and time-consuming task of feature-signal extraction from the pixel-based intensity data obtained by scanning a microarray.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a short DNA polymer 100, called an oligomer, composed of the following subunits: (1) deoxy-adenosine 102; (2) deoxy-thymidine 104; (3) deoxy-cytosine 106; and (4) deoxy-guanosine 108.

Figures 2A-B illustrate the hydrogen bonding between the purine and pyrimidine bases of two anti-parallel DNA strands.

Figure 3 illustrates a short section of a DNA double helix 300 comprising a first strand 302 and a second, anti-parallel strand 304.

Figures 4-7 illustrate the principle of the array-based hybridization assay.

Figures 8A-E illustrate feature-pixel/background-pixel partitioning problems when the partitioning is based on feature-pixel/background-pixel intensity differentials.

Figure 9 is a high-level control-flow diagram of the routine "generateFeatureMask."

Figure 10 illustrates an exemplary pixel-based ROI intensity data set.

Figure 11 shows a preliminary feature mask resulting from an initial partitioning of the pixels in the ROI shown in Figure 10 by the routine "generateFeatureMask."

Figure 12 illustrates the preliminary feature mask initialized in step 904 of Figure 9 after the hole-filling operation of step 906 of Figure 9.

Figure 13 shows the feature mask of Figure 12 following the iterative feature-mask refinement step 912 in Figure 9.

Figure 14 illustrates the feature mask of Figure 13 following a second hole-filling operation.

Figure 15 is a control-flow diagram of the iterative feature-mask refinement step 912 of the routine "generateFeatureMask," illustrated in Figure 9.

Figure 16 shows a Boolean mask indicating intensity outliers, detected during step 1504 of Figure 15, for the ROI shown in Figure 10 and initial feature mask shown in Figure 12.

Figures 17A-B illustrate the probability distributions for signal noise and for pixels intensities with respect to pixel location, respectively.

Figure 18 is a control-flow diagram illustrating a currently anticipated feature extraction program.

Figure 19 is a flow-control diagram of a feature-extraction program incorporating the feature-mask generation method that represents one embodiment of the present invention.

Figure 20 shows an image of a graphical output from a modified feature-extraction program in which the sizes and the locations of features identified in the scanned image of a microarray are graphically displayed to a user.

DETAILED DESCRIPTION OF THE INVENTION

One embodiment of the present invention provides a method and system for discriminating between pixels, in a digital image of a microarray, associated with features and pixels in inter-feature regions of the microarray image referred to as background pixels. In a

first subsection, below, additional information about microarrays is provided. Those readers familiar with microarrays may skip over this first subsection. In a second subsection, embodiments of the present invention are provided through examples, graphical representations, and with reference to several flow-control diagrams. In a third subsection, a C++-like pseudocode implementation of one embodiment of the present invention is provided, to illustrate details omitted from the higher-level discussion in the previous subsection. An alternative embodiment of the present invention is included, in Appendix A.

Additional Information About Molecular Arrays

An array may include any one-, two- or three-dimensional arrangement of addressable regions, or features, each bearing a particular chemical moiety or moieties, such as biopolymers, associated with that region. Any given array substrate may carry one, two, or four or more arrays disposed on a front surface of the substrate. Depending upon the use, any or all of the arrays may be the same or different from one another and each may contain multiple spots or features. A typical array may contain more than ten, more than one hundred, more than one thousand, more ten thousand features, or even more than one hundred thousand features, in an area of less than 20 cm^2 or even less than 10 cm^2. For example, square features may have widths, or round feature may have diameters, in the range from a 10 μm to 1.0 cm. In other embodiments each feature may have a width or diameter in the range of 1.0 μm to 1.0 mm, usually 5.0 μm to 500 μm, and more usually 10 μm to 200 μm. Features other than round or square may have area ranges equivalent to that of circular features with the foregoing diameter ranges. At least some, or all, of the features may be of different compositions (for example, when any repeats of each feature composition are excluded the remaining features may account for at least 5%, 10%, or 20% of the total number of features). Interfeature areas are typically, but not necessarily, present. Interfeature areas generally do not carry probe molecules. Such interfeature areas typically are present where the arrays are formed by processes involving drop deposition of reagents, but may not be present when, for example, photolithographic array fabrication processes are used. When present, interfeature areas can be of various sizes and configurations.

Each array may cover an area of less than 100 cm^2, or even less than 50 cm^2, 10 cm^2 or 1 cm^2. In many embodiments, the substrate carrying the one or more arrays will be shaped generally as a rectangular solid having a length of more than 4 mm and less than 1 m, usually more than 4 mm and less than 600 mm, more usually less than 400 mm; a width of more than 4 mm and less than 1 m, usually less than 500 mm and more usually less than 400 mm; and a thickness of more than 0.01 mm and less than 5.0 mm, usually more than 0.1 mm and less than 2 mm and more usually more than 0.2 and less than 1 mm. Other shapes are possible, as well. With arrays that are read by detecting fluorescence, the substrate may be of a material that emits low fluorescence upon illumination with the excitation light. Additionally in this situation, the substrate may be relatively transparent to reduce the absorption of the incident illuminating laser light and subsequent heating if the focused laser beam travels too slowly over a region. For example, a substrate may transmit at least 20%, or 50% (or even at least 70%, 90%, or 95%), of the illuminating light incident on the front as may be measured across the entire integrated spectrum of such illuminating light or alternatively at 532 nm or 633 nm.

Arrays can be fabricated using drop deposition from pulsejets of either polynucleotide precursor units (such as monomers) in the case of *in situ* fabrication, or the previously obtained polynucleotide. Such methods are described in detail in, for example, US 6,242,266, US 6,232,072, US 6,180,351, US 6,171,797, US 6,323,043, U.S. Patent Application Serial No. 09/302,898 filed April 30, 1999 by Caren et al., and the references cited therein. Other drop deposition methods can be used for fabrication, as previously described herein. Also, instead of drop deposition methods, photolithographic array fabrication methods may be used. Interfeature areas need not be present particularly when the arrays are made by photolithographic methods as described in those patents.

A microarray is typically exposed to a sample including labeled target molecules, or, as mentioned above, to a sample including unlabeled target molecules followed by exposure to labeled molecules that bind to unlabeled target molecules bound to the array, and the array is then read. Reading of the array may be accomplished by illuminating the array and reading the location and intensity of resulting fluorescence at multiple regions on each feature of the array. For example, a scanner may be used for this purpose, which is similar to the AGILENT MICROARRAY SCANNER manufactured by Agilent

Technologies, Palo Alto, CA. Other suitable apparatus and methods are described in U.S. patent applications: Serial No. 10/087447 "Reading Dry Chemical Arrays Through The Substrate" by Corson et al., and in U.S. Patents 6,518,556; 6,486,457; 6,406,849; 6,371,370; 6,355,921; 6,320,196; 6,251,685; and 6,222,664. However, arrays may be read by any other method or apparatus than the foregoing, with other reading methods including other optical techniques, such as detecting chemiluminescent or electroluminescent labels, or electrical techniques, where each feature is provided with an electrode to detect hybridization at that feature in a manner disclosed in US 6,251,685, US 6,221,583 and elsewhere.

A result obtained from reading an array may be used in that form or may be further processed to generate a result such as that obtained by forming conclusions based on the pattern read from the array, such as whether or not a particular target sequence may have been present in the sample, or whether or not a pattern indicates a particular condition of an organism from which the sample came. A result of the reading, whether further processed or not, may be forwarded, such as by communication, to a remote location if desired, and received there for further use, such as for further processing. When one item is indicated as being remote from another, this is referenced that the two items are at least in different buildings, and may be at least one mile, ten miles, or at least one hundred miles apart. Communicating information references transmitting the data representing that information as electrical signals over a suitable communication channel, for example, over a private or public network. Forwarding an item refers to any means of getting the item from one location to the next, whether by physically transporting that item or, in the case of data, physically transporting a medium carrying the data or communicating the data.

As pointed out above, array-based assays can involve other types of biopolymers, synthetic polymers, and other types of chemical entities. A biopolymer is a polymer of one or more types of repeating units. Biopolymers are typically found in biological systems and particularly include polysaccharides, peptides, and polynucleotides, as well as their analogs such as those compounds composed of, or containing, amino acid analogs or non-amino-acid groups, or nucleotide analogs or non-nucleotide groups. This includes polynucleotides in which the conventional backbone has been replaced with a non-naturally occurring or synthetic backbone, and nucleic acids, or synthetic or naturally occurring nucleic-acid analogs, in which one or more of the conventional bases has been

replaced with a natural or synthetic group capable of participating in Watson-Crick-type hydrogen bonding interactions. Polynucleotides include single or multiple-stranded configurations, where one or more of the strands may or may not be completely aligned with another. For example, a biopolymer includes DNA, RNA, oligonucleotides, and PNA and other polynucleotides as described in US 5,948,902 and references cited therein, regardless of the source. An oligonucleotide is a nucleotide multimer of about 10 to 100 nucleotides in length, while a polynucleotide includes a nucleotide multimer having any number of nucleotides.

As an example of a non-nucleic-acid-based microarray, protein antibodies may be attached to features of the array that would bind to soluble labeled antigens in a sample solution. Many other types of chemical assays may be facilitated by array technologies. For example, polysaccharides, glycoproteins, synthetic copolymers, including block copolymers, biopolymer-like polymers with synthetic or derivitized monomers or monomer linkages, and many other types of chemical or biochemical entities may serve as probe and target molecules for array-based analysis. A fundamental principle upon which arrays are based is that of specific recognition, by probe molecules affixed to the array, of target molecules, whether by sequence-mediated binding affinities, binding affinities based on conformational or topological properties of probe and target molecules, or binding affinities based on spatial distribution of electrical charge on the surfaces of target and probe molecules.

Scanning of a microarray by an optical scanning device or radiometric scanning device generally produces a scanned image comprising a rectilinear grid of pixels, with each pixel having a corresponding signal intensity. These signal intensities are processed by an array-data-processing program that analyzes data scanned from an array to produce experimental or diagnostic results which are stored in a computer-readable medium, transferred to an intercommunicating entity via electronic signals, printed in a human-readable format, or otherwise made available for further use. Molecular array experiments can indicate precise gene-expression responses of organisms to drugs, other chemical and biological substances, environmental factors, and other effects. Molecular array experiments can also be used to diagnose disease, for gene sequencing, and for analytical chemistry. Processing of microarray data can produce detailed chemical and biological analyses, disease diagnoses, and other information that can be stored in a computer-readable medium,

transferred to an intercommunicating entity via electronic signals, printed in a human-readable format, or otherwise made available for further use.

Overview of the Present Invention

When a microarray is scanned, data may be collected as a two-dimensional digital image of the microarray, each pixel of which represents the intensity of phosphorescent, fluorescent, chemiluminescent, or radioactive emission from an area of the microarray corresponding to the pixel. A microarray data set may comprise a two-dimensional image or a list of numerical, alphanumerical pixel intensities, or any of many other computer-readable data sets. An initial series of steps employed in processing scanned, digital microarray images includes constructing a regular coordinate system for the digital image of the microarray by which the features within the digital image of the microarray can be indexed and located. For example, when the features are laid out in a periodic, rectilinear pattern, a rectilinear coordinate system is commonly constructed so that the positions of the centers of features lie as closely as possible to intersections between horizontal and vertical gridlines of the rectilinear coordinate system. Then, regions of interest ("ROIs") are computed, based on the initially estimated positions of the features in the coordinate grid, and centroids for the ROIs are computed in order to refine the positions of the features. Once the position of a feature is refined, feature pixels can be differentiated from background pixels within the ROI, and the signal corresponding to the feature can then be computed by integrating the intensity over the feature pixels.

The techniques described above generally rely on discriminating feature pixels from background pixels on the basis of differences in intensity between feature pixels and background pixels. The centroids of ROIs are moments of intensity in a two-dimensional pixel space. Unfortunately, there are many cases where discriminating features based on intensity leads to inaccurate and even grossly inaccurate partitioning of pixels between feature pixels and background pixels, and may lead to an inability to identify feature pixels, especially in the case of low-signal features surrounded by noisy background pixels.

Figures 8A-E illustrate feature-pixel/background-pixel partitioning problems when the partitioning is based on feature-pixel/background-pixel intensity differentials. Figure 8A shows a 25 x 25 pixel region of a digital image of a microarray. Each two-digit

number in the two-dimensional array of numbers in Figure 8A, such as two-digit number "02" 802, represents the intensity associated with a pixel. The intensities are displayed in a two-dimensional array coincident with the two-dimensional array of pixels in the microarray-image region and referenced by x and y coordinates based on an x axis 804 and a y axis 806. Thus, the intensity of the pixel at location (0,0) with respect to the x and y axes is shown as the two-digit number "02" 802 in Figure 8A. The intensity of pixel (24, 24) 808 is represented by the two-digit number "07." The two-digit numbers represent the pixel intensity divided by 100. The pixel intensities shown in the region illustrated in Figure 8A were generated computationally in order to provide a hypothetical example, variants of which are used throughout the subsequent discussion of various embodiments of the present invention. In these subsequent discussions, a number of figures similar to Figure 8A are referenced, and the illustration conventions described above with respect to Figure 8A are used, as well, in these subsequently referenced figures.

Visual inspection of the pixel intensities in the microarray-image region shown in Figure 8A reveals a roughly disk-shaped region with relatively high intensities surrounded by an annulus of intermediate intensities that together represent the image of a feature. Figure 8B shows the region of a hypothetical microarray illustrated in Figure 8A overlaid with two roughly circular line boundaries of the disk-shaped high-intensity and intermediate-intensity portions of the microarray-image region. The highest-intensity portion of the microarray-image region 810 shows intensities varying between 4800 and 5099, since integer division by 100 is employed to produce the two-digit representations of intensities, and the intermediate-intensity annular portion 812 or the microarray-image region surrounding the high-intensity portion contains intensities between 2300 and 2799. The remaining portion of the microarray-image region 814 contains relatively low-intensity pixels, having an average intensity of 1000, and corresponds to the background of the microarray-image region. Thus, as shown in Figures 8A-B, feature pixels in the microarray-image region illustrated in Figures 8A-B can be readily chosen from the microarray-image region by choosing a threshold intensity, and allocating those pixel intensities below the threshold intensity to a background-pixel partition, and allocating those pixels above a threshold pixel intensity to a feature-pixel partition. As described above, currently employed techniques computationally identify an

intensity moment, or centroid, in the subregion and then identify a feature region approximately coincident with the centroid based on intensity thresholding.

Figure 8C shows a second hypothetical region of a microarray. This second hypothetical microarray-image region is identical to the microarray-image region illustrated in Figures 8A-B, except that the average intensity of the high-intensity, central feature pixels is only 2000, rather than 5000. Note that, although the feature can still be visually distinguished by carefully examining the two-dimensional array of pixel intensities, the task of visually discriminating feature pixels from background pixels is significantly more difficult. Figure 8D shows the high-intensity, central feature portion of the microarray-image region, illustrated in Figure 8C, outlined. Figure 8E shows a microarray-image region similar to those shown in Figures 8A-D, with exception that the average intensities of feature pixels within the high-intensity, central portion of the feature is only 1500. Thus, in Figure 8E, the high-intensity, central feature pixels have an average intensity only 1.5 times that of the background pixels. Note that it is quite difficult to distinguish feature pixels from background pixels based on intensity alone. Note also that these hypothetical examples are, in many ways, best-case types of examples, in that the feature region is regularly shaped and relatively uniform in intensity. In actual microarray data, features may be asymmetrical, may contain highly non-uniform intensities due to a large number of different possible procedural, experimental, and instrumental errors and instabilities, and may be substantially offset from their expected positions within the general grid-like, regular pattern in which features are deposited. All of these effects can lead to difficulties in employing the currently used, intensity-based feature-pixel/background-pixel partitioning methods described above.

Embodiments of the present invention are described, in overview, in this subsection with reference to control-flow diagrams provided as Figures 9, 15, and 17-18 and discussed. In addition, an exemplary, pixel-based region-of-interest ("ROI") data set and a number of feature masks are included as Figures 10-14 and 16, and discussed in this subsection. Following this overview subsection, a next subsection provides a detailed, C++-like pseudocode implementation of one embodiment of the present invention. In the described embodiments, both channels of a 2-channel scan of a microarray are passed as pixel-based signal intensity input data sets to the described method. In one embodiment, a simple average of the signals in each channel for each feature is computed. In alternative

embodiments, the signals for each channel may be separately processed, and feature masks prepared for each channel. The data sets may be separately processed through feature extraction, or the feature masks for each channel later combined to generate a combination feature mask. The raw intensity signal is generally modified to give greater weight to lower intensity signals. Modification may be as simple as taking the logarithm of the signal, or may involve a more complex weighting scheme. The alternative implementation provided in Appendix A uses a non-linear weighting scheme, embodied in the transform "Alaw companding," that maintains a linear range for low-intensity data, but compresses the intensity ranges for higher intensity data.

Figure 9 is a high-level control-flow diagram of the routine "generateFeatureMask." The routine "generateFeatureMask" receives pixel-based intensity data for an ROI and generates a corresponding feature mask that indicates which of the pixels in the ROI correspond to the image of a feature, and which pixels in the ROI correspond to background. In Step 902, the routine "generateFeatureMask" receives pixel-based intensity data for an ROI extracted from the complete pixel-based intensity data within an entire scanned image of a microarray. In other words, the routine "generateFeatureMask" would be invoked to generate a feature mask for each ROI extracted from the scanned image of a microarray. In alternative embodiments, the routine "generateFeatureMask" may be implemented to generate a single, multi-feature-containing feature mask for an entire scanned image of a microarray, either sequentially or in parallel.

Figure 10 illustrates an exemplary pixel-based ROI intensity data set. In Figure 10, the ROI 1002 is a two-dimensional array of the natural logarithms of the intensities measured for pixels within the ROI in the scanned image of the ROI. As can be seen in Figure 10, a circular area 1004 of relatively high intensities corresponding to feature pixels is surrounded by an area 1006 of relatively lower intensities corresponding to background pixels. However, there are two smaller areas 1008 and 1010 of relatively high-intensity pixels at two of the corners of the ROI. Various aberrant high-intensity regions, such as regions 1008 and 1010, frequently occur in ROI intensity data sets as a result of microarray contamination, defects in microarray manufacture, instrumental error in microarray scanners, contamination of the surface of the microarray or of the sample solutions to which the microarray is exposed, and as a result of many other events. It is a goal of feature-mask

generation to accurately identify those pixels within the pixel-based ROI intensity data set that correspond to the image of a feature and to mask out high-intensity pixels, such as those in regions 1008 and 1010 in Figure 10, that do not correspond to feature pixels. In addition, there may be various, isolated intensity-outlying pixels, both in the background and feature regions of an ROI, that also need to be identified and taken into consideration during signal extraction from pixel-intensity data.

Returning to Figure 9, the routine "generateFeatureMask," in step 904, undertakes an initial partitioning of the pixels within the ROI into a set of feature pixels and a set of background pixels. The details of one example of initial partitioning are provided in the following subsection as a C++-like pseudocode routine. In the subsequently described initial partitioning method, pixels are assigned to the feature-pixel set and to the background-pixel set based on the intensities of the pixels with respect to a threshold intensity. Figure 11 shows a preliminary feature mask resulting from the initial partitioning of the pixels in the ROI shown in Figure 10 by the routine "generateFeatureMask." Comparing Figure 11 to Figure 10, it is apparent that the initial partitioning has classified pixels within the two non-feature, high-intensity regions 1008 and 1010, along with the central, highest-intensity pixels of the feature region 1004 as feature pixels, represented by the Boolean value "1" in the feature mask 1102. The Boolean value "0" represents a background pixel in the feature mask. Note that the feature mask 1102 includes a Boolean value for every pixel in the pixel-based ROI intensity data set 1002 shown in Figure 10. Note also that the small, core feature region 1004 in the feature mask 1102 includes some "0" values. These values, generally surrounded predominantly by "1" values, are referred to as holes within the Boolean representation of feature pixels. Oppositely valued holes may also appear within the background. These holes are dealt with in step 906 of the routine "generateFeatureMask." In step 906, a simple hole-filling algorithm, described in detail in the following subsection, is employed to detect holes within feature and background regions and repair them. Figure 12 illustrates the preliminary feature mask initialized in step 904 after the hole-filling operation of step 906. Note that, in general, the holes have been largely repaired, both in the region corresponding to the feature 1202 as well as in the non-feature, high-intensity regions 1204 and 1206.

Returning to Figure 9, the routine "generateFeatureMask," in step 908, next evaluates the initial feature mask, as modified by the hole-filling operation, to determine

whether or not the initial feature mask is acceptable. Various different acceptability criteria may be employed. In the described embodiment, as discussed in the following subsection, a hypothetical radius for the feature identified by Boolean "1" values in the feature mask is computed based on the area of the feature mask containing Boolean "1" values. This feature radius is compared with a putative feature radius independently determined by a gridding-type routine to determine whether or not the initial feature mask appears to indicate a feature of size comparable to the putative feature size obtained either by gridding methods that determine and extract ROIs from scanned images of microarrays, from a knowledge of the microarray manufacture process, or from a combination of such sources. Gridding methods involve computing a best fit for a rectilinear coordinate system that places features at grid intersections. If the initial mask is okay, as determined in step 910, then the routine "generateFeatureMask" directly undertakes an iterative feature-mask refinement, in step 912. Otherwise, the routine "generateFeatureMask," in step 914, repartitions the pixels into feature pixels and background pixels by another, alternative partitioning method, prior to undertaking iterative feature-mask refinement in step 912.

In the iterative, feature-mask refinement step, 912, the routine "generateFeatureMask" carries out a Bayesian-probability-based method for repartitioning pixels based on mathematical models for the probability distributions of feature and background pixel intensity noise and mathematical models for probability distributions for feature-pixel and background-pixel intensities with respect to the distances of pixels from a putative location of the center of the feature within the ROI, as independently determined by a gridding method. Figure 13 shows the feature mask following the iterative feature-mask refinement step 912 in Figure 9. As can be seen by comparing Figure 13 to Figure 12, the iterative feature-mask refinement step has eliminated the non-feature, high-intensity pixels in regions 1204 and 1206 of Figure 12 from the feature-pixel partition, and has expanded the relatively small, core feature area (1202 in Figure 12) to a much larger feature area 1302 that closely corresponds to the area (1004 in Figure 10) of the feature based on pixel-intensity values as shown in Figure 10. However, the iterative feature-mask refinement step 912 has also resulted in the reappearance of holes within the feature area 1302, such as hole 1304. Therefore, in step 916, the routine "generateFeatureMask" again undertakes a hole-filling

operation, resulting in a final feature mask. Figure 14 shows the final feature mask following the hole-filling operation of step 916.

Next, in step 918, the routine "generateFeatureMask" determines the centroid and feature radius of the feature represented in the final feature mask, an example of which is shown in Figure 14. Then, in step 920, the routine "generateFeatureMask" checks the calculated centroid and radius of the feature-mask feature with the putative center and putative radius of the feature generated earlier during preparation of the ROI received in step 902. If the calculated centroid and radius correspond sufficiently closely to the putative centroid and putative radius, as determined in step 922, then the routine "generateFeatureMask" updates the putative centroid and putative radius with the calculated centroid and calculated radius of the feature as represented by the feature mask, in step 924, and returns a true Boolean value in step 926. Otherwise, the routine "generateFeatureMask" returns a false Boolean value in step 928 to indicate that the generated feature mask differs considerably from the putative feature characteristics.

Figure 15 is a control-flow diagram of the iterative feature-mask refinement routine invoked in step 912 of the routine "generateFeatureMask," illustrated in Figure 9. In a first step 1502, the iterative feature-mask refinement routine computes various statistics based on the intensity data in the ROI intensity data set and based on the partitioning of pixels into feature pixels and background pixels, as represented by the feature mask. Next, in step 1504, intensity-outlying pixels are identified so that they can be ignored in subsequent calculations. Figure 16 shows a Boolean mask indicating intensity outliers detected during step 1504 for the ROI shown in Figure 10 and initial feature mask shown in Figure 12. The intensity outliers are designated by Boolean value "1" in the outlier mask shown in Figure 16. Once the intensity outliers are identified, the statistics are again computed, in step 1506, without considering the intensity outliers identified in step 1504. Next, a Bayesian-probability-based classification of the pixels is carried out in step 1508. This step is discussed, in greater detail in this subsection, below, as well as in the following subsection. Finally, in step 1510, the iterative feature-mask refinement routine determines whether or not feature-mask refinement has converged, in a described embodiment by determining whether or not only a relatively small number of pixels have been reclassified from feature pixels to background pixels and vice-versa in the current Bayesian-probability-based repartitioning of

step 1508. If the iterative feature-mask refinement has converged, as determined in step 1512, then the iterative feature-mask refinement routine returns, in step 1514. Otherwise, control flows back to step 1502 for a subsequent iteration.

The Bayesian-probability-based repartitioning, undertaken in step 1508 of the iterative feature-mask refinement routine illustrated in Figure 15, is based on the following probability models for each channel:

$$P(F/i,x) = \frac{P(F,i,x)}{P(i,x)} = \frac{P(i/x,F)P(F,x)}{P(i,x)} = \frac{P(i/x,F)P(F/x)P(x)}{P(i,x)}$$

$$P(B/i,x) = \frac{P(B,i,x)}{P(i,x)} = \frac{P(i/x,B)P(B,x)}{P(i,x)} = \frac{P(i/x,B)P(B/x)P(x)}{P(i,x)}$$

where

$P(F/i,x)$ is the probability that a pixel is a feature pixel given the pixel's intensity i and location x;

$P(B/i,x)$ is the probability that a pixel is a background pixel given the pixel's intensity i and location x;

$P(F,i,x)$ is the probability of a pixel being a feature pixel having intensity i and location x;

$P(B,i,x)$ is the probability of a pixel being a background pixel having intensity i and location x;

$P(i,x)$ is the probability of a pixel having intensity i and location x;

$P(i/x,F)$ is the probability of a pixel having intensity i given the pixel's location x and the fact that the pixel is a feature pixel;

$P(i/x,B)$ is the probability of a pixel having intensity i given the pixel's location x and the fact that the pixel is a background pixel;

$P(F,x)$ is the probability of a pixel being a feature pixel and having a location x;

$P(B,x)$ is the probability of a pixel being a background pixel and having a location x;

$P(F/x)$ is the prior probability that a pixel belongs to feature given its location x;

$P(B/x)$ is the prior probability that a pixel belongs to the background given its location x; and

$P(x)$ is the probability that a pixel has location x.

The probability distribution of pixel intensities due to noise for both feature and background pixels are modeled as Gaussian distributions:

$$P(i/x,F) = \frac{1}{\sqrt{2\pi}\sigma_F} e^{\frac{-(i-\mu_F)^2}{2\sigma_F^2}}$$

$$P(i/x,B) = \frac{1}{\sqrt{2\pi}\sigma_B} e^{\frac{-(i-\mu_B)^2}{2\sigma_B^2}}$$

where

σ_F is the standard deviation of the feature-pixel intensities;

σ_F^2 is the variance of the feature-pixel intensities;

μ_F is the mean of the feature-pixel intensities;

σ_B is the standard deviation of the background-pixel intensities;

σ_B^2 is the variance of the background -pixel intensities;

μ_B is the mean of the background -pixel intensities.

The prior probabilities are modeled as functions of the distance of a pixel from the putative feature centroid:

$$P(F/x) = f_F(|x_{putative} - x|)$$

$$P(B/x) = f_B(|x_{putative} - x|)$$

The exact functions f_F and f_B are provided in the C++-like pseudocode implementation, in the following subsection and can be replaced by more complex functions requiring more computation time

Figures 17A-B illustrate the general probability distributions for the prior probabilities. In Figure 17A, two Gaussian intensity distributions are shown for prior probabilities $P(i/x,F)$ and $P(i/x,B)$. In Figure 17B, the general tendencies of the probability distributions for prior probabilities $P(F/x)$ and $P(B/x)$ are shown with respect to distance of a pixel from the putative feature centroid. In the C++-like pseudocode implementation, piecewise linear probability distributions models are employed, to simplify calculation.

Up to this step, all of the above mentioned calculations are performed for two channels separately. The suffix red/green for each equation is avoided to minimize unnecessary complexity in notation. A joint probability distribution of the two channels is then computed. In the described embodiment, the two channels are considered independent and hence the joint probability density function is the product of the probability density functions for the red and green channels.

The Bayesian-probability calculations and calculation of the joint probability density function are simplified by observing that, for repartitioning, one only needs to determine the relative probabilities of a pixel being a feature or background pixel, given its intensity and location, as shown below:

$$\log(\frac{P(F/i,x)}{P(B/i,x)}) = \log(\frac{P(i/x,F)P(F,x)}{P(i/x,B)P(B,x)}) = \log(\frac{P(i/x,F)P(F/x)}{P(i/x,B)P(B/x)})$$
$$= \log(\frac{\sigma_B}{\sigma_F}) - \frac{(i-\mu_F)^2}{2\sigma_F^2} + \log(F/x) + \frac{(i-\mu_B)^2}{2\sigma_B^2} - \log(B/x)$$

If the log ratio of this joint probability of a pixel being a feature pixel to the joint probability of a pixel being a background is calculated, as described in the above equation, then the criteria for assigning a pixel to the feature-pixel category is as follows:

$$\log(\frac{P(F/i,x)}{P(B/i,x)}) \geq 0$$

or

$$\log(P(F/i,x)) - \log(P(B/i,x)) \geq 0$$

while the criteria for assigning a pixel to the background-pixel category is as follows:

$$\log(\frac{P(F/i,x)}{P(B/i,x)}) < 0$$

or

$$\log(P(F/i,x)) - \log(P(B/i,x)) < 0$$

where all the probability density functions are the joint probability density functions. By computing the log of the ratio, one can avoid the fairly expensive computation of the exponential term and sequential double precision multiplications.

The generation of feature masks by the above-described method provides a number of advantages during analysis of microarray data. First, generation of such feature masks is a robust and accurate method that can be performed during feature extraction to identify the features within the ROIs. In addition, because the feature-mask generation method described above relies only on putative feature sizes and positions and the intensity data within an ROI intensity data set, feature masks for ROI's can be easily generated in advance of complex feature-extraction calculations in order to provide a preview of the detected features, including detected feature sizes and locations, to users prior to undertaking feature extraction.

Figure 18 is a control-flow diagram illustrating a currently anticipated feature extraction program. In step 1802, the feature-extraction program receives a scanned image from a microarray, including pixel-intensities corresponding to pixels within the scanned image. Next, in step 1804, a gridding technique is employed in order to determine putative ROIs and feature locations and sizes. Then, in steps 1805-1806, complex and time-consuming image-processing and post-image-processing methods are carried out in order to generate the final output data, including the ratios of the intensities above the background signal, normalized for red and green channels, for each feature within the scanned image of the microarray. In step 1808, the results of feature extraction are displayed to a user, following which, in step 1810, user input is received based on displayed results. If the user indicates that the feature extraction has appeared to have been successfully performed, as determined in step 1812, then the feature extraction completes in step 1814. Otherwise, the feature-extraction program receives corrections regarding computed spot centroids and radii from a user, in step 1816. Feature extraction may then be repeated.

Figure 19 is a flow-control diagram of a feature-extraction program incorporating the feature-mask generation method that represents one embodiment of the present invention. Many of the steps in Figure 19 are identical to the steps of the currently anticipated feature-extraction program, illustrated in Figure 17, and are not described again. Note, however, that after the gridding step 1904, the modified feature-extraction program undertakes generation of a feature mask, in step 1906. Based on the generated feature mask, the modified feature-extraction program, in step 1908, displays a preview of the detected feature locations and sizes within the ROIs obtained by the gridding step 1904. Next, user input is solicited, in step 1910. If the user finds the calculated feature positions and sizes, displayed in step 1908, to be acceptable, as determined in step 1912, then image processing and image post processing are undertaken in steps 1913-1914. Otherwise, the modified feature-extraction program solicits corrections from the user, in step 1916, and control flows back to step 1908, where the corrections are displayed. Thus generation of a feature mask by the above-described method that represents one embodiment of the present invention allows for a preview and iterative correction of initial feature locations and sizes to be undertaken prior to feature extraction. The modified feature-extraction program therefore allows a user to preview and modify the initial feature positions and sizes prior to the computationally intensive and time-consuming post image processing leading to a final result. This preview mode provided by the modified feature extraction program can considerably shorten the overall use of computing resources and shorten the time for successful feature extraction, in certain cases, by soliciting user input immediately following the image processing step and before the analysis.

Figure 20 shows an image of a graphical output from a modified feature-extraction program in which the sizes and the locations of features identified in the scanned image of a microarray are graphically displayed to a user. A user can then individually modify feature locations and sizes based on the visual display of the image processing results, so that, once feature extraction is undertaken, the user is confident that the best initial positions and sizes of features will be used.

A C++-Like Pseudocode Implementation of One Embodiment of the Present Invention

In this subsection, a straightforward C++-like pseudocode implementation of one embodiment of the present invention is provided for illustration only. A commercial implementation is provided in Appendix A. This pseudocode implementation is provided for clarity and illustration, and employs programming constructs and methodologies that most simply illustrate the feature-mask generation method that represents one embodiment of the present invention. As with any software implementation, greater efficiencies may be achieved by employing less clear, but more efficient techniques, well known to ordinarily skilled software developers. For example, nested *for*-loop iteration of elements in two-dimensional arrays may be more efficiently handled by special assembly-language iterators. As another example, various techniques may be employed to optimize arithmetic calculations to avoid the expense of floating-point operations without sacrificing the accuracy of calculated results.

The described method that represents one embodiment of the present invention generates a Boolean-valued feature mask from a pixel-based ROI intensity data set, as described in the previous subsection. The ROI data set is represented, in the following pseudocode implementation, by an instance of the class "scanned ROI" that follows:

```
 1 class scannedROI
 2 {
 3
 4     public:
 5         int pixel(int x, int y);
 6         int width();
 7         int height();
 8         int total(); // width * height
 9         getX();
10         int getY();
11         int getRad();
12         scannedROI();
13 };
```

The function-member implementations of the public methods of the above class "scannedROI" are not provided, as they are outside the scope of the present discussion. An instance of the class "scannedROI" can be accessed through the listed public function members on lines 5-12, above, that include: (1) "pixel," a function member that returns the intensity of the pixel at two-dimensional-array coordinates "*x,y*"; (2) "width, " a function

member that returns the width of the two-dimensional-array representation of the ROI, in pixels; (3) "height, " a function member that returns the height of the two-dimensional-array representation of the ROI, in pixels; (4) "total, " a function member that returns the total number of pixels in the ROI; (5) "getX, " a function member that returns the putative *x*-coordinate for the center of the feature within the ROI; (6) "getY, " a function member that returns the *y* coordinate for the putative center of the feature within the ROI; (7) "getRad, " a function member that returns the putative radius of the feature within the ROI; and (8) a constructor for the class "scannedROI." The above-listed set of function members provides all the information needed by the feature-mask generation routine in order to prepare a Boolean-valued feature mask indicating the positions of the feature pixels within the ROI.

Next, a number of integer constants are provided:

```
1 const int FlipLim = 26;
2 const int IterationLim = 10;
3 const int MainLoopIterationLim = 10;
4 const int OutlierIterationLim = 10;
```

These constants are representative threshold values that indicate convergence or that limit the number of iterations of loops within function members, as discussed below.

Next, a declaration for the class "featureFinder" that implements one embodiment of the present invention is provided:

```
1 class featureFinder
2 {
3     private:
4             FILE* s;
5             scannedROI*  roi;
6
7             double bVar, fVar;
8             double bSD, fSD;
9             double fAv, bAv;
10            int curX, curY, curRad;
11
12            bool featureMask[HEIGHT][WIDTH];
13            bool tMask[HEIGHT][WIDTH];
14            void KMeans();
15            void holeFill();
16            bool andNeighborhood(int x, int y, bool t);
17            bool orNeighborhood(int x, int y, bool t);
18            void computeStatistics();
```

```
19              bool initialAcceptance();
20              void alternateInitialClassification();
21              void rejectOutliers();
22              bool BayesianClassify();
23              void findCentroid();
24
25      public:
26              void newROI(scannedROI*   r);
27              bool generateFeatureMask(int & x, int & y, int & rad);
28              void clear();
29              featureFinder(FILE* st);
30 };
```

The class "featureFinder" includes the following private data members: (1) "s," a pointer to an output stream through which results are output to files; (2) "roi," a pointer to a region of interest from which a feature mask is generated; (3) "fVar," "bVar," "fSD," "bSD," "fAv" and "bAv," declared above on lines 7-9, which store the variances, standard deviations, and means for the intensity distributions of feature pixels and background pixels; (4) "curX," "curY," and "curRad," declared above on line 10, that store a feature position and feature radius calculated from a generated feature mask; (5) "featureMask," a two-dimensional Boolean-valued representation of the ROI, as described in a previous subsection; and (6) "tMask," a second Boolean-valued representation of the ROI, used for storing intermediate results and for storing indications of intensity outliers. The class "featureFinder" includes the following private function members: (1) "KMeans," declared above on line 14, a function member that performs an initial partitioning of ROI pixels into a set of feature pixels and a set of background pixels; (2) "holeFill," declared above on line 15, a function member that performs a hole-filling operation on a generated feature mask; (3) "andNeighborhood," declared above on line 16, a function member that calculates the Boolean AND of the feature-mask values corresponding to pixels adjoining a specified pixel along edges of the specified pixel; (4) "orNeighborhood," a function member similar to the previously described function member that computes the Boolean OR of the Boolean values corresponding to neighboring pixels of the specified pixel; (5) "computeStatistics," declared above on line 18, a function member that computes the statistical values stored in the data members declared on lines 7-9, above; (6) "initialAcceptance," a function-member that returns a Boolean value indicating whether or not an initially generated feature mask represents an acceptable first pass at identifying feature pixels within the ROI; (7) "alternateInitialClassification," a function member that

performs an alternative initial partitioning of pixels into feature pixels and background pixels; (8) "rejectOutliers," a function member that identifies and removes intensity outliers from consideration in subsequent calculations; (9) "BayesianClassify," declared above on line 22, a function-member that carries out an iterative Bayesian-probability-based refinement of an initially generated feature mask; and (10) "findCentroid," a function member that determines the coordinates of the center of the feature based on the generated feature mask.

The public function members for the class "featureFinder" include: (1) "newROI," declared above on line 26, which receives a pointer to an instance of the class "scannedROI" representing a pixel-based ROI intensity data set from which a feature mask can be prepared; (2) "generateFeatureMask," declared above on line 27, a function member that generates a feature mask, as described in the previous subsection, and returns calculated location and radius of the feature identified in the feature mask via the variable parameters "x" "y" and "rad"; and (3) a constructor.

An implementation of the private function member KMeans is next provided:

```
 1 void featureFinder::KMeans()
 2 {
 3     int i, j, k, t;
 4     double sumLow, sumHigh;
 5     int low = 1000000, high = -1;
 6     int flips = FlipLim + 1;
 7     int pivot;
 8     bool feature;
 9
10     for (i = 0; i < roi->width(); i++)
11     {
12         for (j = 0; j < roi->height(); j++)
13         {
14             t = roi->pixel(i,j);
15             if (t > high) high = t;
16             if (t < low) low = t;
17         }
18     }
19     pivot = (high  + low) / 2;
20     for (k = 0; k < IterationLim; k++)
21     {
22         sumLow = 0;
23         low = 0;
24         sumHigh = 0;
25         high = 0;
26         flips = 0;
27
```

```
28              for (i = 0; i < roi->width(); i++)
29              {
30                      for (j = 0; j < roi->height(); j++)
31                      {
32                              t = roi->pixel(i,j);
33                              feature = t >= pivot;
34                              if (featureMask[i][j] != feature)
35                              {
36                                      featureMask[i][j] = feature;
37                                      flips++;
38                              }
39                              if (feature)
40                              {
41                                      high++;
42                                      sumHigh += t;
43                              }
44                              else
45                              {
46                                      low++;
47                                      sumLow += t;
48                              }
49                      }
50              }
51              if (flips < FlipLim) break;
52              pivot = int(((sumHigh / high) + (sumLow / low)) / 2);
53      }
54 }
```

The function member "KMeans" iteratively partitions pixels into a set of feature pixels and a set of background pixels. First, in the nested *for*-loop on lines 10-18, the pixels of the ROI are scanned to determine the highest and lowest intensity values within the ROI. The highest intensity value is stored in the variable "high," while the lowest intensity value detected in the ROI is stored in the variable "low." Then, on line 19, the variable "pivot" is initialized to be the intensity value intermediate between the highest and lowest intensity values detected within the ROI. Next, in the *for*-loop of lines 20-53, the pixels are iteratively partitioned until either a number of iterations equal to the constant "IterationLim" have been executed, or until the number of pixels reclassified from feature to background and from background to feature falls below the threshold value "FlipLim," as detected on line 51. In the nested *for*-loops of lines 28-50, the intensity of each pixel in the ROI is compared to the current value of the variable "pivot," on line 33. If the intensity value of the pixel is greater than or equal to the current value of the variable "pivot," the pixel is classified as a feature pixel, and otherwise is classified as a background pixel. If, in the current iteration of the *for*-loop of lines 20-53, the

classification of a pixel changes, then the value stored in the variable "flips" is incremented, on line 37. In addition, a running total of the number of feature pixels and background pixels, and a running sum of the intensities of the feature pixels and background pixels, is maintained in the variables "high," "low," "sumHigh," and "sumLow," updated on lines 39-48. If convergence is not reached in the current iteration of the *for*-loop of lines 20-53, as detected on line 51, then a new value for the variable "pivot" is calculated on line 52 as an intensity value intermediate between the mean of the feature pixels and the mean intensity of the background pixels.

Next, an implementation of the function member "andNeighborhood" is provided. A detailed annotation of this implementation is not provided, as implementation is quite straightforward. The function member "andNeighborhood" computes the Boolean AND of the Boolean values corresponding to pixels adjacent to a pixel specified by the input arguments "x" and "y." Input parameter "t" specifies whether the Boolean and operation is conducted on the two-dimensional Boolean array "feature mask" or on the two-dimensional Boolean array "tMask." An implementation for the similar function member "orNeighborhood" is not provided, as that implementation is almost identical with implementation of the function member "andNeighborhood," with the exception that all "&&" operators are replaced with "||" operators. It should also be noted that more efficient implementations of "andNeighborhood" and "orNeighborhood" are possible, in which the treatment of special-case corner and edge pixels in the implementation, provided below, can be avoided.

```
bool featureFinder::andNeighborhood(int x, int y, bool t)
{
    if (t)
    {
        if (x > 0 && x < roi->width()-1 && y > 0 && y < roi->height() -1)
        {
            return (tMask[x][y-1] && tMask[x][y+1] &&
                tMask[x-1][y] && tMask[x+1][y]);
        }
        else if (x == 0)
        {
            if (y == 0)
            {
                return (tMask[0][1] && tMask[1][0]);
            }
```

```
                else if (y == roi->height() - 1)
                {
                        return (tMask[0][roi->height() - 2] &&
                                tMask[1][roi->height() - 1]);
                }
                else
                {
                        return (tMask[0][y-1] && tMask[0][y+1] &&
                                tMask[1][y]);
                }
        }
        else if (x == roi->width() - 1)
        {
                if (y == 0)
                {
                        return (tMask[roi->width()-2][0] &&
                                tMask[roi->width()-1][1]);
                }
                else if (y == roi->height() - 1)
                {
                        return (tMask[roi->width()-2][roi->height()-1] &&
                                tMask[roi->width()-1][roi->height()-2]);
                }
                else
                {
                        return (tMask[roi->width()-1][y-1] &&
                                tMask[roi->width()-1][y+1] &&
                                tMask[roi->width()-2][y]);
                }
        }
        else if (y == 0)
        {
                return (tMask[x-1][0] && tMask[x+1][0] && tMask[x][1]);
        }
        else
        {
                return (tMask[x-1][roi->height()-1] &&
                        tMask[x+1][roi->height()-1] &&
                        tMask[x][roi->height()-2]);
        }
}
else
{
        if (x > 0 && x < roi->width()-1 && y > 0 && y < roi->height() -1)
        {
                return (featureMask[x][y-1] && featureMask[x][y+1] &&
                        featureMask[x-1][y] && featureMask[x+1][y]);
        }
        else if (x == 0)
        {
                if (y == 0)
                {
```

```
                        return (featureMask[0][1] && featureMask[1][0]);
                }
                else if (y == roi->height() - 1)
                {
                        return (featureMask[0][roi->height() - 2] &&
                                featureMask[1][roi->height() - 1]);
                }
                else
                {
                        return (featureMask[0][y-1] && featureMask[0][y+1] &&
                                featureMask[1][y]);
                }
        }
        else if (x == roi->width() - 1)
        {
                if (y == 0)
                {
                        return (featureMask[roi->width()-2][0] &&
                                featureMask[roi->width()-1][1]);
                }
                else if (y == roi->height() - 1)
                {
                        return (featureMask[roi->width()-2][roi->height()-1] &&
                                featureMask[roi->width()-1][roi->height()-2]);
                }
                else
                {
                        return (featureMask[roi->width()-1][y-1] &&
                                featureMask[roi->width()-1][y+1] &&
                                featureMask[roi->width()-2][y]);
                }
        }
        else if (y == 0)
        {
                return (featureMask[x-1][0] && featureMask[x+1][0] &&
                        featureMask[x][1]);
        }
        else
        {
                return (featureMask[x-1][roi->height()-1] &&
                        featureMask[x+1][roi->height()-1] &&
                        featureMask[x][roi->height()-2]);
        }
    }
}
```

Next, an implementation for the function member "holeFill" is provided:

```
1 void featureFinder::holeFill()
2 {
```

```
3     int i, j;
4
5     for (i = 0; i < roi->width(); i++)
6     {
7           for (j = 0; j < roi->height(); j++)
8           {
9                 tMask[i][j] = featureMask[i][j] || orNeighborhood(i,j, false);
10          }
11    }
12    for (i = 0; i < roi->width(); i++)
13    {
14          for (j = 0; j < roi->height(); j++)
15          {
16                featureMask[i][j] = tMask[i][j] && andNeighborhood(i,j, true);
17          }
18    }
19    for (i = 0; i < roi->width(); i++)
20    {
21          for (j = 0; j < roi->height(); j++)
22          {
23                if (!featureMask[i][j])
24                      featureMask[i][j] = andNeighborhood(i,j, false);
25                else featureMask[i][j] = orNeighborhood(i,j, false);
26                tMask[i][j] = false;
27          }
28    }
29 }
```

The function member "holeFill" successfully applies a dilation operation, in the nested *for*-loops of lines 5-11, an erosion operation in the nested *for*-loops of lines 12-18, and a final hole-filling operation in the nested *for*-loops of lines 19-28. The dilation operation results are placed into the two-dimensional Boolean array "tMask," on line 9, which results are then used in the erosion operation, the results of which are, in turn, migrated back to the two-dimensional array "featureMask." The dilation operation tends to flip Boolean values "0" to Boolean values "1" for pixels in the neighborhood of feature pixels, the erosion operation expands areas of Boolean-value "0" within the feature mask, and the hole-filling operation detects and removes feature pixels surrounded by background pixels and background pixels surrounded by feature pixels.

Next, an implementation for the function member "initialAcceptance" is provided:

```
1 bool featureFinder::initialAcceptance()
```

```
 2 {
 3     double fNum = 0;
 4     double area_based_rad;
 5     double diff;
 6
 7     int i, j;
 8
 9     for (i = 0; i < roi->width(); i++)
10     {
11             for (j = 0; j < roi->height(); j++)
12             {
13                     if (featureMask[i][j]) fNum++;
14             }
15     }
16     area_based_rad = sqrt(fNum / 3.141);
17     diff = (rad - area_based_rad) / roi->getRad();
18     if (diff < 0) diff = -diff;
19     return (diff < 0.25);
20 }
```

The function member "initialAcceptance" determines a theoretical, area-based radius by taking the square root of the number of feature pixels, divided by π, on line 16. If the difference between the area-based radius and the putative feature radius reported by the instance of the class "scannedROI" is less than 25 percent of the putative feature radius, then the function member "initialAcceptance" returns a Boolean value "TRUE" indicating that the initially generated feature mask is acceptable. Otherwise, a Boolean value "FALSE" is returned. Note that the exact threshold for acceptability may differ from the difference of less than 25 percent of the putative feature radius in alternative embodiments.

Next, an implementation of the function member "alternateInitialClassification" is provided below:

```
1 void featureFinder::alternateInitialClassification()
2 {
3     int i, j;
4     double root, dis1, dis2;
5
6     for (i=0; i < roi->width(); i++)
7     {
8       for (j=0; j < roi->height(); j++)
9       {
10              dis1 = (i - roi->getX()) * (i - roi->getX());
11              dis2 = (j - roi->getY()) * (j - roi->getY());
12              root = sqrt(dis1 + dis2);
13              if (root <= roi->getRad()) featureMask[i][j] = true;
```

```
14                else featureMask[i][j] = false;
15        }
16      }
17 }
```

The function member "alternateInitialClassification" simply generates an initial feature mask with Boolean-value "1" values corresponding to the pixels within the putative feature as described by the feature location and feature radius reported by an instance of the class "scannedROI." This is accomplished by considering each pixel, in the nested *for*-loops of lines 6-16, and determining whether the distance of the pixel from the putatitve feature center is less than or equal to the putative radius of the feature, as reported by the instance of the class "scannedROI."

Next, an implementation of the function member "computeStatistics" is provided below:

```
 1 void featureFinder::computeStatistics()
 2 {
 3      int i, j;
 4      int bNum = 0;
 5      int fNum = 0;
 6      double bSum = 0;
 7      double fSum = 0;
 8      double bSum2 = 0;
 9      double fSum2 = 0;
10
11      for (i = 0; i < roi->width(); i++)
12      {
13              for (j = 0; j < roi->height(); j++)
14              {
15                      if (!tMask[i][j])
16                      {
17                              if (featureMask[i][j])
18                              {
19                                      fSum += roi->pixel(i,j);
20                                      fSum2 += roi->pixel(i,j) * roi->pixel(i,j);
21                                      fNum++;
22                              }
23                              else
24                              {
25                                      bSum += roi->pixel(i,j);
26                                      bSum2 += roi->pixel(i,j) * roi->pixel(i,j);
27                                      bNum++;
28                              }
29                      }
```

```
30              }
31      }
32      fAv = fSum / fNum;
33      bAv = bSum / bNum;
34      fVar = (fSum2/fNum) - (fAv * fAv);
35      bVar = (bSum2/bNum) - (bAv * bAv);
36      bSD = sqrt(bVar);
37      fSD = sqrt(fVar);
38 }
```

The function member "computeStatistics" traverses all of the pixels within the ROI in the nested *for*-loops of lines 11-31, accumulating a sum of feature-pixel intensities, a sum of the squares of feature-pixel intensities, and a number of feature pixels, and similar quantities for background pixels. From these values, the means, variances, and standard deviations for the feature pixels and background pixels are calculated on lines 32-37.

Next, an implementation for the function member "rejectOutliers" is provided, below:

```
 1 void featureFinder::rejectOutliers()
 2 {
 3      int initial_fLow = int (fAv - (3 * fSD));
 4      int initial_fHigh = int (fAv + (3 * fSD));
 5      int initial_bLow = int (bAv - (3 * bSD));
 6      int initial_bHigh = int (bAv + (3 * bSD));
 7
 8      double bOut = roi->total();
 9      double fOut = roi->total();
10      int t;
11
12      int i, j, k = 0;
13
14      while (k < OutlierIterationLim && (bOut + fOut) > (0.25 * roi->total()))
15      {
16              bOut = 0;
17              fOut = 0;
18
19              for (i = 0; i < roi->width(); i++)
20              {
21                      for (j = 0; j < roi->height(); j++)
22                      {
23                              t = roi->pixel(i, j);
24                              if (featureMask[i][j])
25                              {
26                                      if (t < initial_fLow || t > initial_fHigh)
27                                      {
28                                              tMask[i][j] = true;
```

```
29                                      fOut += 1;
30                                  }
31                                  else tMask[i][j] = false;
32                              }
33                              else
34                              {
35                                  if (t < initial_bLow || t > initial_bHigh)
36                                  {
37                                      tMask[i][j] = true;
38                                      bOut += 1;
39                                  }
40                                  else tMask[i][j] = false;
41                              }
42                          }
43                  }
44                  initial_fLow -= (initial_fHigh - initial_fLow) / 3;
45                  initial_fHigh += (initial_fHigh - initial_fLow) / 3;
46                  initial_bLow -= (initial_bHigh - initial_bLow) / 3;
47                  initial_bHigh += (initial_bHigh - initial_bLow) / 3;
48                  k++;
49      }
50 }
```

The function member "rejectOutliers" iteratively identifies intensity outliers in the *while*-loop of lines 14-49. Initially, outlier rejection is accepted if less than 25 percent of the total number of pixels are rejected as intensity outliers. However, if a larger percentage of pixels are rejected, then outlier rejection is again performed by widening the intensity criteria for non-intensity-outlying pixels. Note that the two-dimensional Boolean array "tMask" is used for indicating intensity outliers with a Boolean value "1." The range of acceptable intensities for feature pixels is stored in the two variables "initial_fLow" and "initial_fHigh," and the range for acceptable background intensities as stored in the variables "initial_bLow" and "initial_bHigh." Thus, in the nested *for*-loops of lines 19-43, all pixels are considered, and those pixels for which intensities lie outside of the acceptable range are flagged as intensity outliers. If, following consideration of all pixels, more than 25 percent of the total pixels are classified as outliers, the ranges of acceptable intensities are broadened on lines 44-47.

Next, an implementation of the function member "BayesianClassify" is provided, below:

```
1 bool featureFinder::BayesianClassify()
2 {
3     double Pi_xF, Pi_xB, PB_x, PF_x;
```

```
4
5     double recipFVar = log(1 / (sqrt(6.282) * fSD));
6     double recipBVar = log(1 / (sqrt(6.282) * bSD));
7
8     double root, dis1, dis2;
9     double lowRad, highRad, intermediateRad;
10
11    int numFlipped = 0;
12    bool t;
13    int i, j;
14
15    for (i = 0; i < roi->width(); i++)
16    {
17        for (j = 0; j < roi->height(); j++)
18        {
19            if (!tMask[i][j])
20            {
21                t = featureMask[i][j];
22                Pi_xF = recipFVar –
23                    ((roi->pixel(i, j) - fAv) *
24                     (roi->pixel(i, j) - fAv) / fVar);
25                Pi_xB = recipBVar -
26                    ((roi->pixel(i, j) - bAv) *
27                     (roi->pixel(i, j) - bAv) / bVar);
28
29                dis1 = (i - roi->getX()) * (i - roi->getX());
30                dis2 = (j - roi->getY()) * (j - roi->getY());
31                root = sqrt(dis1 + dis2);
32
33                lowRad = 0.5 * roi->getRad();
34                highRad = 1.5 * roi->getRad();
35                intermediateRad = highRad - lowRad;
36
37                if (root <= lowRad)
38                {
39                    PB_x = -100000;
40                    PF_x = 0;
41                }
42                else if (root >= highRad)
43                {
44                    PF_x = -100000;
45                    PB_x = 0;
46                }
47                else
48                {
49                    PB_x = log((root - lowRad) / intermediateRad);
50                    PF_x = log((highRad - root) / intermediateRad);
51                }
52
53                if(t)
54                {
55                    if ((Pi_xF + PF_x) - (Pi_xB + PB_x) < 0)
```

```
56                                  {
57                                      featureMask[i][j] = false;
58                                      numFlipped++;
59                                  }
60                              }
61                              else
62                              {
63                                  if ((Pi_xF + PF_x) - (Pi_xB + PB_x) >= 0)
64                                  {
65                                      featureMask[i][j] = true;
66                                      numFlipped++;
67                                  }
68                              }
69                          }
70                      }
71              }
72              return (numFlipped < roi->total() / 100);
73 }
```

The function member "BayesianClassify" computes the Bayesian probability for each pixel as being a feature pixel and as being a background pixel. When the ratio of the Bayesian probability of a pixel being a feature pixel provided by the probability of the feature being a background pixel is greater than or equal to one, the pixel is classified as a feature pixel. The detailed mathematical models for the Bayesian probability calculation are discussed in the previous subsection. In the nested *for*-loops of lines 15-74, the Bayesian probabilities for each non-outlying pixel are calculated, and the pixel is classified according to those calculated probabilities. First, the probability of the observed intensity of the pixel given the location of the pixel and the fact of the pixel as a feature pixel is calculated on lines 22-24. A corresponding probability of the observed intensity of the pixel, given the location of the pixel and the fact that the pixel is a background pixel, is calculated on lines 25-27. Then, the probabilities of the pixel being a background pixel, given its location, and being a feature pixel, given its location, are calculated on lines 33-51. The logarithms of the probabilities, rather than the probabilities are calculated and maintained by the function member "BayesianClassify." Logarithmic values are more easily and efficiently manipulated. Finally, if the pixel was originally classified as a feature pixel, as determined on line 53, then, if the ratio of the pixel being a feature pixel versus the probability of the feature being a background pixel is greater than or equal to one or, in other words, the log ratio is greater than or equal to zero, then the feature remains classified as a feature pixel. Otherwise, the feature is classified

as a background pixel, and the variable "numFlipped" is incremented in order to maintain a count of the total number of pixels reclassified by the function member "BayesianClassify." Similarly, if the pixel is initially classified as a background pixel, the value is flipped if the log ratio is greater than or equal to zero. Finally, on line 72, the function member "BayesianClassify" returns a Boolean value indicating whether or not the number of pixels reclassified is less than one percent of the total number of pixels. If so, then the Boolean value "TRUE" is returned. In alternative embodiments, a different convergence criteria may be used.

Next, an implementation of the function member "findCentroid" is provided, below:

```
 1 void featureFinder::findCentroid()
 2 {
 3     int i, j, num;
 4     int xsum, ysum, sum;
 5     double area_based_rad;
 6
 7     xsum = 0;
 8     ysum = 0;
 9     sum = 0;
10     num = 0;
11
12     for (i=0; i < roi->width(); i++)
13         for (j=0; j < roi->height(); j++)
14         {
15             if (featureMask[i][j])
16             {
17                 xsum += i * roi->pixel(i,j);
18                 ysum += j * roi->pixel(i,j);
19                 sum += roi->pixel(i,j);
20                 num++;
21             }
22         }
23     curX = xsum / sum;
24     curY = ysum / sum;
25     area_based_rad = sqrt(num / 3.141);
26     curRad = int(area_based_rad);
27 }
```

The function member "findCentroid" computes the first moment of intensity for the pixels identified as feature pixels with respect to the x coordinate axis and the y coordinate axis in

order to determine the *x,y* coordinates of the center of the feature. The radius of the feature is calculated based on the total area of the feature mask represented by the Boolean value "1."

A simple implementation of the function member "newROI" is provided, below, without additional annotation:

```
1 void featureFinder::newROI(scannedROI* r)
2 {
3     roi = r;
4 }
```

Next, an implementation of the function member "generateFeatureMask" is provided, below:

```
1 bool featureFinder::generateFeatureMask(int & x, int & y, int & rad)
2 {
3     int k;
4     int xDiff, yDiff, rDiff;
5
6     KMeans();
7     holeFill();
8     if (!initialAcceptance())
9         alternateInitialClassification();
10    while (k < MainLoopIterationLim)
11    {
12        computeStatistics();
13        rejectOutliers();
14        computeStatistics();
15        if (BayesianClassify())  break;
16        k++;
17    }
18    holeFill();
19    findCentroid();
20    xDiff = roi->getX() - curX;
21    yDiff = roi->getY() - curY;
22    rDiff = roi->getRad() - curRad;
23    if (xDiff < 0) xDiff = -xDiff;
24    if (yDiff < 0) yDiff = -yDiff;
25    if (rDiff < 0) rDiff = -rDiff;
26
27    if (xDiff < (roi->width() / 10) &&
28        yDiff < (roi->height() / 10) &&
29        rDiff < roi->getRad() / 3)
30        {
31            x = curX;
32            y = curY;
33            rad = curRad;
```

```
34                      return true;
35              }
36      else return false;
37 }
```

The function member "generateFeatureMask" closely corresponds to the control-flow diagrams discussed in the previous subsection. First, on line 6, the function member "KMeans" is called to initially classify pixels as being either feature pixels or background pixels. Next, the function member "holeFill" is called, on line 7, in order to fill holes within the initially generated feature mask. If the initially generated feature mask is not acceptable, as determined by calling the function member "initialAcceptance" on line 8, then an alternative initial feature mask is generated by calling the function member "alternateInitialclassification" on line 9. In the *while*-loop of lines 10-17, the function member "generateFeatureMask" iteratively computes statistics for feature and background pixels, on line 12, rejects intensity outliers, on line 13, re-computes statistics, on line 14, and reclassifies pixels based on Bayesian probabilities calculated for each pixel on line 15. The *while*-loop terminates either when the repartitioning of pixels by the function member "BayesianClassify" indicates convergence of the reclassification, or when the maximum of iterations, represented by the constant "MainLoopIterationLim," has been executed. Next, on line 18, the function member "holeFill" is called in order to fill any holes arising from the reclassification of pixels. Then, on line 19, the function member "findCentroid" is called in order to compute the centroid and radius of the feature as represented by the generated feature mask. When the computed location and size of the feature is close to the putative feature location and size, as determined on lines 20-30, then the calculated feature position and size is reported back to the caller of the function member "generateFeatureMask" via the variable arguments.

Finally, for completeness, a constructor for the class "featureFinder" is provided without further annotation, below:

```
1 featureFinder::featureFinder(FILE* st)
2 {
3       int i, j;
4
5       s = st;
6       for (i = 0; i < roi->width(); i++)
```

```
    {
        for (j = 0; j < roi->height(); j++)
        {
            featureMask[i][j] = false;
        }
    }
}
```

A simple, example routine in which a feature mask is generated from a pixel-based ROI data intensity set is next provided, without further annotation:

```
int main(int argc, char* argv[])
{

    scannedROI sROI;
    featureFinder f(stream);
    int x, y, rad;

    f.newROI(&sROI);
    f.generateFeatureMask(x, y, rad);
}
```

Although the present invention has been described in terms of a particular embodiment, it is not intended that the invention be limited to this embodiment. Modifications within the spirit of the invention will be apparent to those skilled in the art. For example, an almost limitless number of software implementations for the method of feature-mask generation can be crafted by ordinarily skilled software developers using different modularization, control structures, data structures, programming languages, and various other different characteristics and techniques. In the described embodiment, various limits to the number of loop iterations and constants that specify convergence criteria are employed, but, in alternative embodiments, constants may have different values or entirely different means for constraining the overall calculation that may be employed. As discussed above, generation of feature masks by the method of the present invention may be employed within feature-extraction packages both to provide a preview mode in which users can interactively review and change feature positions and sizes prior to the time-consuming and computationally expensive feature extraction steps as well as to accurately identify feature sizes and locations within scanned images within the feature extraction steps. In the described embodiment, the method of the present invention is focused on a single pixel-based

ROI intensity data set, but the method can be expanded for employment on multi-signal intensity data sets, including one or more regions of interest. The classification may be stored in Boolean arrays or bit-map arrays, or in alternative data structures.

The foregoing description, for purposes of explanation, used specific nomenclature to provide a thorough understanding of the invention. However, it will be apparent to one skilled in the art that the specific details are not required in order to practice the invention. The foregoing descriptions of specific embodiments of the present invention are presented for purpose of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously many modifications and variations are possible in view of the above teachings. The embodiments are shown and described in order to best explain the principles of the invention and its practical applications, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents: